CMSC 671 (Introduction to AI) – Fall 2018

Homework 4 (98 points)

Turnin: Blackboard.

Please Parts I, II, III, and questions 14-17 together as a single PDF file named *lastname*_hw4.pdf (named after the person who submits), and part IV as zipped up .py files.

You are encouraged to work on this homework assignment **in your project groups** (no other groups). If you do so, you only need to **turn in one solution**, with everyone's name in the file. **Remember, if you work in a group, you MUST work on the problems *AS* a group, not split up the work.**

PART I. LEARNING IN THE WILD (8 PTS.)

Assignment: Consider the problem faced by a robot trying to figure out which of the objects that it can see are *manipulable* (small enough to to be picked up and handled). *(Maximum: 300 words)*

1. Explain how this problem fits into the general learning model.
2. Describe the percepts (sensor inputs) and actions of the robot.
3. Describe the types of learning the robot must do.
4. Describe the subfunctions the robot is trying to learn in terms of inputs, outputs, and available training data.

PART II. DECISION TREE LEARNING (30 PTS.)

Consider the training examples shown in the following table of data instances for a binary classification problem with three attributes.

Instance	*a1*	*a2*	*a3*	*Class*
X_1	T	T	1.0	+
X_2	T	T	6.0	+
X_3	T	F	5.0	-
X_4	F	F	4.0	+
X_5	F	T	7.0	-
X_6	F	T	3.0	-
X_7	F	F	8.0	-
X_8	T	F	7.0	+
X_9	F	T	5.0	-

Assignment: Answer the following questions about Table 1, showing all work.

5. What is the entropy of this collection of training examples?
6. What are the information gains of *a1* and *a2* relative to these training examples?
7. For *a3* (which is continuous), compute the information gain for every possible split.
8. What is the best split (among *a1*, *a2*, and *a3*) according to the information gain?
9. What is the best split (between *a1* and *a2*) according to the classification error rate?

PART III. RESOLUTION AND FORMAL LOGIC (30 PTS.)

We are going to represent the space of gardening in predicate calculus and first-order logic. Use the following predicates:

fertilized (*x*)	*x* is fertilized.
in-season(*x*)	*x* is planted in the correct season.
rooted(*x*)	*x* is well rooted in the ground.
hardy(*x*)	*x* is hardy (robust, sturdy, hard to kill).
die(*x*, *t*)	*x* died during planting *t*.
survive(*x*, *t*)	*x* survived planting *t*.
watered (*x*)	*x* is watered well.
native(*x*)	*x* is native to the area.

Where arguments *x* implicitly take the domain of all plants, and arguments *t* takes the domain of all plantings. (That is, you don't need to include predicates like plant(*x*).)

Knowledge Representation (18 pts)

Assignment:

10. Represent the following knowledge base in first-order logic. (8 pts)
    a) Everything that is hardy, watered, and fertilized will be well-rooted.
    b) Everything that is well-rooted will survive a planting if it is in season.
    c) A plant survives a planting if and only if they don't die.
    d) Every native plant is hardy.
    e) If a planting isn't in season, every plant will die in the planting.
    f) My inkberry is a native plant.
    g) My holly survived the fall planting.
    h) I fertilize my inkberry.
11. Convert the KB to conjunctive normal form. (Hint: you will need to define three constants, in addition to the predicates above and variables *x* and *t*.) (8 points)
12. Express the negation of this statement: (inkberry) → survive(inkberry, fall-planting) in conjunctive normal form. (2 pts)

Proving (12 points)

Assignment: Prove that (inkberry) → survive(inkberry, fall-planting).

13. Adding the negated goal to the KB and using resolution refutation to prove that it is true. You may show your proof as a series of sentences to be added to the KB or as a proof tree. In either case, you must clearly show which sentences are resolved to produce each new sentence, and what the unifier is for each resolution step.

PART IV. FORGING A PATH (30 PTS.)

The General Idea

We are extending the code we have written previously to solve path-finding problems. As before, the agent will need to try to find a path between the start and goal positions. There are two major changes to consider:

- ***Walls.*** We are introducing an additional type of block, "Wall," which cannot be moved through.
- ***Observability.*** The agent can only perceive the eight squares directly surrounding it. The agent also cannot see through walls (in Figure 1, (0, 7) will never be observable.)
- ***Optimality.*** The agent will be looking for a *good* path, but not necessarily optimal.

Figure 1: an explored 7x7 map

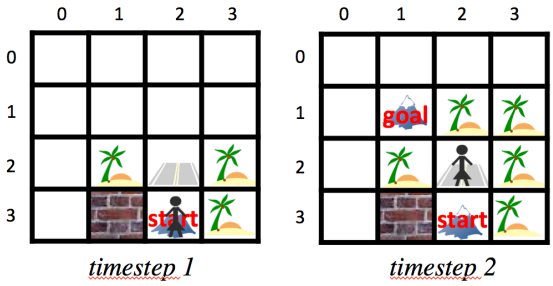

Figure 2: the first two hill-climbing timesteps

You should extend your existing code to read in and solve puzzles of the type described. **While the complete board will be passed in, the *agent* does not know the complete board, and does not know where the goal state is until finding it** (see Figure 2). This means that you will have to use some form of local search (we'll use beam search) to choose a path; furthermore, the agent must interleave exploring the space with expanding the search tree.

Write a program to read in and solve puzzles of the type described above. The puzzle may be of any size, will always be square, and will have a non-random selection of path, sand, wall, and mountain cells. The goal of the puzzle is to move your agent from the starting cell to the goal cell. S and G may be any cell.

Details

You may assume:

- **The size of the square is between 7x7 and 50x50.**
- **The agent may only move one square at a time (not diagonally).**
- **There will always be some non-blocked path.**
- **Agents may backtrack. (They will have to, in fact.)**
- **The landscape is static (no cells will ever change once observed).**

Assignment: Write a function called `solve` which takes a matrix (the problem) and two tuples (start and goal), and tries to find a path from the start state to the goal state. Your solver should take these values, and **return** (not print) a string containing a list of moves. Moves should be represented by the capital letters N,S,E,W for north, south, east, and west moves. Note that because the space is not fully observable, the agent will backtrack occasionally; an agent in (5,5) in Figure 1 might well move west, then east.

Please don't print anything inside the function.

Implement your solution to this problem as: **local beam search**, using any heuristic you like. (Hint: make sure your heuristic is abstracted well, as it will change in the project.) Because we are using beam search and not looking for an optimal solution, your code should run *much* faster than your A* implementation.

Hints:

The biggest difference is that the agent must interleave exploring the space with expanding the search tree; this means you can't simply return the best path found, but must return the path the agent actually traverses.

Assignment: Write up answers to the following in your PDF:

14. What heuristic did you use for choosing what path to explore next?
15. What size beams did you experiment with? What beam size did you use, and why?
16. What was the easiest thing about coding this up? The hardest?
17. How many times did your group meet in solving this? How well did it go?